Отчет по практической работе.
Базы данных и SQLAlchemy.

Выполнил: Матафонов Денис, РИ-380013.

В данном отчете приведены решения задач из презентации по SQLAlchemy с конференции PyCon 2013.

Часть 1. Engine Usage.

В первой лекции показывается, как создать базу данных из alchemy и обращаться к нему при помощи sql-запросов.

Задание такое:

```
+-----------------------------------------------------------------------------+
| *** Exercises ***                                                           |
+-----------------------------------------------------------------------------+
| Assuming this table:                                                        |
|                                                                             |
|     CREATE TABLE employee (                                                 |
|         emp_id INTEGER PRIMARY KEY,                                         |
|         emp_name VARCHAR(30)                                                |
|     }                                                                       |
|                                                                             |
| And using the "engine.execute()" method to invoke a statement:             |
|                                                                             |
| 1. Execute an INSERT statement that will insert the row with emp_name='dilbert'.  |
|    The primary key column can be omitted so that it is generated automatically.   |
|                                                                             |
| 2. SELECT all rows from the employee table.                                 |
+---------------------------------------------------------------- (13 / 13) --+
```

Необходимо добавить в таблицу работников запись про Дилберта, первичный ключ создастся автоматически. Затем необходимо выбрать все строки в таблице.

```
engine.execute("insert into employee (emp_name) values
(:emp_name)", emp_name='dilbert')
```

```
>>> engine.execute("insert into employee (emp_name) values (:emp_name)", emp_name='dilbert')
[SQL]: insert into employee (emp_name) values (:emp_name)
[SQL]: {'emp_name': 'dilbert'}
[SQL]: COMMIT
```

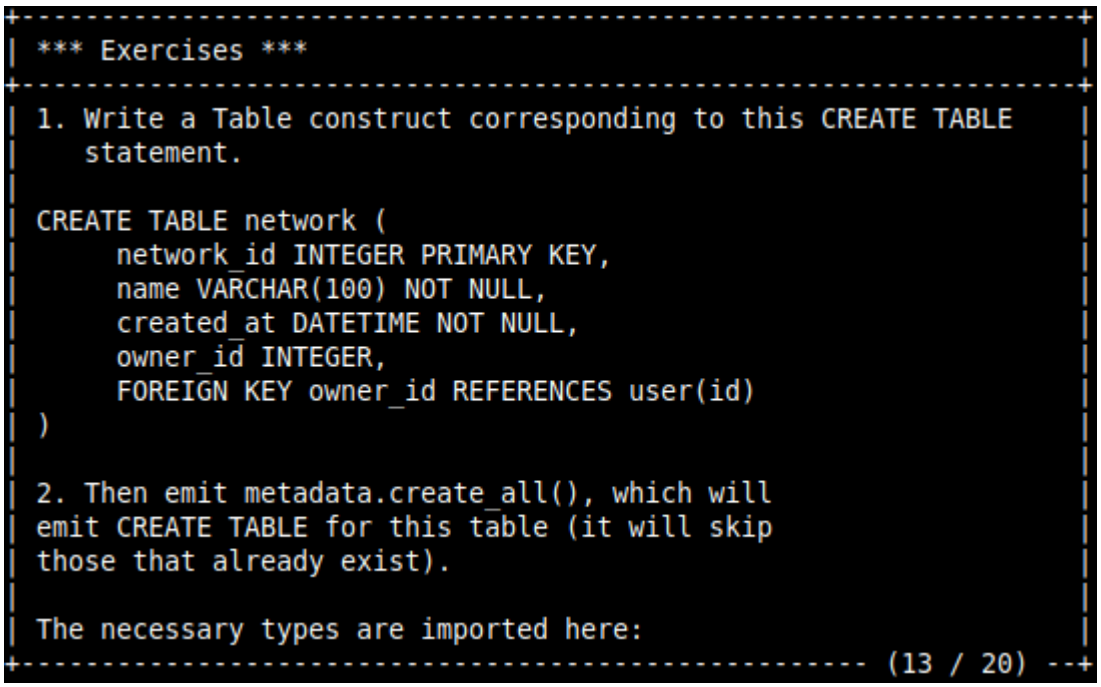
```
res = engine.execute("select * from emloyee")
res.fetchall()
```

```
>>> res = engine.execute("select * from employee")
[SQL]: select * from employee
[SQL]: ()
>>> res.fetchall()
[(1, 'ed'), (2, 'jack'), (3, 'fred'), (4, 'wendy'), (5, 'mary'), (6, 'dilbert')]
```

Действительно, запесь про Дилберта добавлена.

2. Metadata

В этой презентации речь идет об объектах, которыми может быть представлена база данных в Python.



Необходимо написать конструктор таблицы, соответсвующей данной схеме, и создать таблицу с помощью create_all().

```
network_table = Table('network', metadata,
                Column('network_id', Integer, primary_key=True),
                Column('name', String(100), nullable=False),
                Column('created_at', DateTime, nullable=False),
                Column('owner_id', Integer, ForeignKey('user.id'))
        )

engine = create_engine("sqlite://")
metadata.create_all(engine)
```

```
[SQL]:
CREATE TABLE network (
        network_id INTEGER NOT NULL,
        name VARCHAR(100) NOT NULL,
        created_at DATETIME NOT NULL,
        owner_id INTEGER,
        PRIMARY KEY (network_id),
        FOREIGN KEY(owner_id) REFERENCES user (id)
)


[SQL]: ()
[SQL]: COMMIT
```

Таблица создана и помещена в базу sqlite.

Во второй части лекции рассматривается рефлексия полученных метаданных, т. е. можно «достать» структуру таблиц обратно.

```
+-------------------------------------------------------------------------+
| *** Exercises ***                                                       |
+-------------------------------------------------------------------------+
|                                                                         |
| 1. Using 'metadata2', reflect the "network" table in the same way       |
|    we just did 'user', then display the columns (or bonus, display      |
|    just the column names)                                               |
|                                                                         |
| 2. Using "inspector", print a list of all table names that              |
|    include a column called "story_id"                                   |
+------------------------------------------------------- (20 / 20) --+
```

В первом задании необходимо получить структуру таблицы «network» и отобразить её столбцы. Во втором — напечатать список таблиц, имеющих столбец story_id, при помощи инспектора.

```
network_reflected = Table('network', metadata2, autoload=True,
autoload_with=engine)
```

```
[SQL]: PRAGMA table_info("network")
[SQL]: ()
[SQL]: PRAGMA foreign_key_list("network")
[SQL]: ()
[SQL]: PRAGMA index_list("network")
[SQL]: ()
```

```
print(network_reflected.c)
```

```
['network.network_id', 'network.name', 'network.created_at', 'network.owner_id']
```

```
inspector.get_columns('story')
```

```
[{'name': 'story_id', 'type': INTEGER(), 'nullable': False, 'default': None, 'autoincrement': True, 'primary_key': 1}, {'name': 'version_id', 'type': INTEGER(), 'nullabl
e': False, 'default': None, 'autoincrement': True, 'primary_key': 2}, {'name': 'headline', 'type': VARCHAR(length=100), 'nullable': False, 'default': None, 'autoincremen
t': True, 'primary_key': 0}, {'name': 'body', 'type': TEXT(), 'nullable': True, 'default': None, 'autoincrement': True, 'primary_key': 0}]
```

Чтобы получить только названия таблиц:

```
for table in inspector.get_table_names():
     for column in inspector.get_columns(table):
             if column["name"] == "story_id":
                 print(table)
                 break
```

```
[SQL]: PRAGMA table_info("fancy")
[SQL]: ()
[SQL]: PRAGMA table_info("published")
[SQL]: ()
published
[SQL]: PRAGMA table_info("story")
[SQL]: ()
story
[SQL]: PRAGMA table_info("user")
[SQL]: ()
```

3. SQL-expressions.

В этой презентации показано, как получать SQL-выражения при помощи абстракций SQLAlchemy.

```
+-------------------------------------------------------------------+
| *** Exercises ***                                                 |
+-------------------------------------------------------------------+
| Produce these expressions using "user_table.c.fullname",          |
| "user_table.c.id", and "user_table.c.username":                   |
|                                                                   |
| 1. user.fullname = 'ed'                                           |
|                                                                   |
| 2. user.fullname = 'ed' AND user.id > 5                           |
|                                                                   |
| 3. user.username = 'edward' OR (user.fullname = 'ed' AND user.id > 5)  |
+------------------------------------------------------------ (18 / 46) --+
```

В этом упражнений нужно произвести SQL-выражения для следующих записей:
1. Для user с именем ed;
2. Для user с именем ed и id больше 5;
3. Для user с именем пользователя edward или условиями пункта 2.

```
exp1 = user_table.c.fullname == 'ed'
print(exp1)
res1 = exp1.compile().params
print(res1)
```

```
>>> exp1 = user_table.c.fullname == 'ed'
>>> print(exp1)
"user".fullname = :fullname_1
>>> res1 = exp1.compile().params
>>> print(res1)
{'fullname_1': 'ed'}
```

Используем стандартные логические операнды:

```
exp2 = ((user_table.c.fullname == 'ed') & (user_table.c.id > 5))
print(exp2)
res2 = exp2.compile().params
print(res2)
```

```
>>> exp2 = ((user_table.c.fullname == 'ed') & (user_table.c.id > 5))
>>> print(exp2)
"user".fullname = :fullname_1 AND "user".id > :id_1
>>> res2 = exp2.compile().params
>>> print(res2)
```

А здесь — функции из SQLAlchemy or_ и and_:

```
exp2 = or_(
   (user_table.c.username == 'edward'),
   and_(user_table.c.id > 5, user_table.c.fullname == 'ed'))
print(exp2)
res2 = exp2.compile().params
print(res2)
```

```
>>> exp2 = or_(
...     (user_table.c.username == 'edward'),
...     and_(user_table.c.id > 5, user_table.c.fullname == 'ed'))
>>> print(exp2)
"user".username = :username_1 OR "user".id > :id_1 AND "user".fullname = :fullname_1
>>> res2 = exp2.compile().params
>>> print(res2)
{'username_1': 'edward', 'id_1': 5, 'fullname_1': 'ed'}
>>>
```

```
+-------------------------------------------------------------------------------+
| *** Exercises ***                                                             |
+-------------------------------------------------------------------------------+
| 1. use user_table.insert() and "r = conn.execute()" to emit this              |
| statement:                                                                    |
|                                                                               |
| INSERT INTO user (username, fullname) VALUES ('dilbert', 'Dilbert Jones')     |
|                                                                               |
| 2. What is the value of 'user.id' for the above INSERT statement?             |
|                                                                               |
| 3. Using "select([user_table])", execute this SELECT:                        |
|                                                                               |
| SELECT id, username, fullname FROM user WHERE username = 'wendy' OR           |
|   username = 'dilbert' ORDER BY fullname                                      |
+------------------------------------------------------------------ (27 / 46) --+
```

В этом задании необходимо:
1. Добавить запись о Дилберте Джонсе с помощью insert;
2. Узнать, какое будет сгенерировано id для этой записи;
3. Сформировать select-запрос по приведенным условиям.

```
conn.execute(user_table.insert().\
         values(username='dilbert', fullname='Dilbert Jones'))
```

```
>>> conn.execute(user_table.insert().values(username='dilbert', fullname='Dilbert Jones'))
[SQL]: INSERT INTO user (username, fullname) VALUES (?, ?)
[SQL]: ('dilbert', 'Dilbert Jones')
[SQL]: COMMIT
<sqlalchemy.engine.result.ResultProxy object at 0xb65965ec>
```

Чтобы узнать, какое id получил Дилберт, сделаем select по его имени и выберем столбец с id.

```
select_dilbert_id = select([user_table.c.id]).\
                where(user_table.c.fullname == 'Dilbert Jones')
conn.execute(select_dilbert_id).fetchall()
```

```
>>> select_dilbert_id = select([user_table.c.id]).where(user_table.c.fullname == 'Dilbert Jones')
>>> conn.execute(select_dilbert_id).fetchall()
[SQL]: SELECT user.id
FROM user
WHERE user.fullname = ?
[SQL]: ('Dilbert Jones',)
[(4,)]
```

Запись про Дилберта имеет, таким образом, id=4. Итретье подзадание:

```
select_custom = select([user_table.c.id, user_table.c.username, user_table.c.fullname]).\
                    where(or_(
                        user_table.c.username == 'wendy',
                        user_table.c.username == 'dilbert')).\
                    order_by(user_table.c.fullname)
conn.execute(select_custom).fetchall()
```

```
>>> select_custom = select([user_table.c.id, user_table.c.username, user_table.c.fullname]).\
...                                     where(or_(
...                                             user_table.c.username == 'wendy',
...                                             user_table.c.username == 'dilbert')).\
...                                     order_by(user_table.c.fullname)
>>> conn.execute(select_custom).fetchall()
[SQL]: SELECT user.id, user.username, user.fullname
FROM user
WHERE user.username = ? OR user.username = ? ORDER BY user.fullname
[SQL]: ('wendy', 'dilbert')
[(4, 'dilbert', 'Dilbert Jones'), (3, 'wendy', 'Wendy Weathersmith')]
```

Следующая часть презентации посвящена многотабличности и связям.

```
+-------------------------------------------------------------------+
| *** Exercises ***                                                 |
+-------------------------------------------------------------------+
| Produce this SELECT:                                              |
|                                                                   |
| SELECT fullname, email_address FROM user JOIN address             |
|   ON user.id = address.user_id WHERE username='ed'                |
|   ORDER BY email_address                                          |
+------------------------------------------------- (38 / 46) --+
```

Создадим объект обединения таблиц, а потом применим к нему селект.

```
join_obj = user_table.join(address_table, user_table.c.id == address_table.c.user_id)
select_ex = select([user_table.c.fullname, address_table.c.email_address]).\
        select_from(join_obj).\
        where(user_table.c.username == 'ed').\
        order_by(address_table.c.email_address)
conn.execute(select_ex).fetchall()
```

```
>>> select_ex = select([user_table.c.fullname, address_table.c.email_address]).\
...             select_from(join_obj).\
...             where(user_table.c.username == 'ed').\
...             order_by(address_table.c.email_address)
>>> conn.execute(select_ex).fetchall()
[SQL]: SELECT user.fullname, address.email_address
FROM user JOIN address ON user.id = address.user_id
WHERE user.username = ? ORDER BY address.email_address
[SQL]: ('ed',)
[('Ed Jones', 'ed@ed.com'), ('Ed Jones', 'ed@gmail.com')]
```

Действительно, получено два отсортированных почтовых адреса Эда.

Далее рассматриваются операции обновления и удаления.

```
+-------------------------------------------------------------------+
| *** Exercises ***                                                 |
+-------------------------------------------------------------------+
| 1. Execute this UPDATE - keep the "result" that's returned        |
|                                                                   |
|    UPDATE user SET fullname='Ed Jones' where username='ed'        |
|                                                                   |
| 2. how many rows did the above statement update?                  |
|                                                                   |
| 3. Tricky bonus!  Combine update() along with select().as_scalar()|
|    to execute this UPDATE:                                        |
|                                                                   |
|    UPDATE user SET fullname=fullname ||                           |
|        (select email_address FROM address WHERE user_id=user.id) |
|       WHERE username IN ('jack', 'wendy')                         |
+------------------------------------------------------- (46 / 46) --+
```

Необходимо обновить определеные записи о пользователях и затем узнать, скольких записей это коснулось. В последнем упражнении предлагается выполнить обновление при помощи функции as_scalar.

```
update_stmt = user_table.update().\
                    values(fullname="Ed Jones").\
                    where(user_table.c.username == "ed")
result = conn.execute(update_stmt)
```

```
>>> update_stmt = user_table.update().\
...                     values(fullname="Ed Jones").\
...                     where(user_table.c.username == "ed")
>>> result = conn.execute(update_stmt)
[SQL]: UPDATE user SET fullname=? WHERE user.username = ?
[SQL]: ('Ed Jones', 'ed')
[SQL]: COMMIT
```

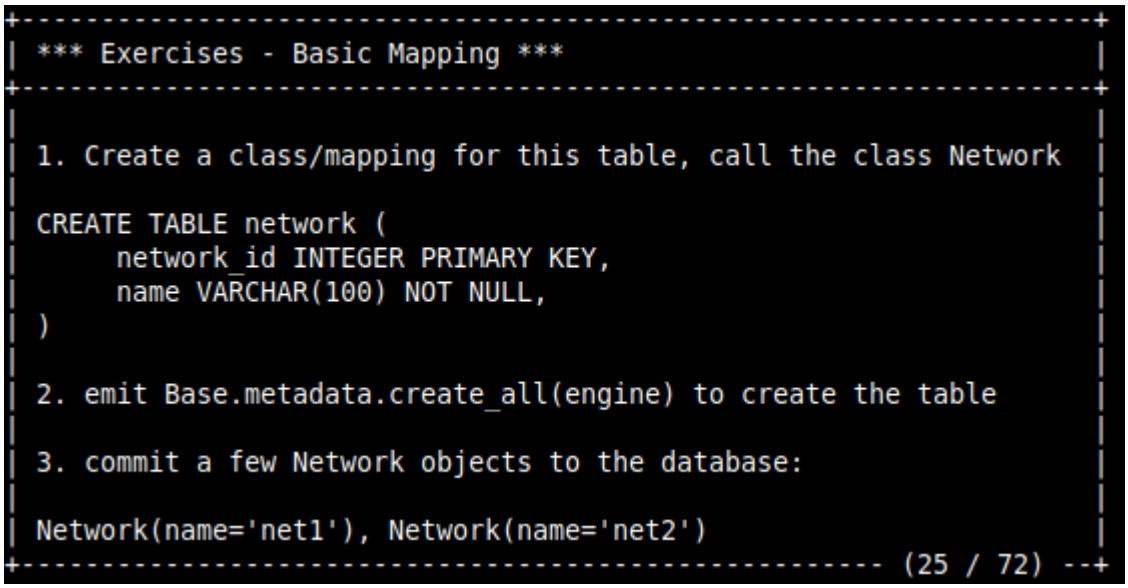
```
result.rowcount
```

```
>>> result.rowcount
1
```

```
email_address_sel = select([address_table.c.email_address]).\
                where(user_table.c.id == address_table.c.user_id)
updater = user_table.update().\
             values(or_(user_table.c.fullname, email_address_sel.as_scalar())).\
             where(user_table.c.username.in_(['jack', 'wendy']))
result = conn.execute(updater)
```

4. ORM

В этой презентации рассматривается использование ORM SQLAlchemy — модели, сопоставляющей объекты Python и таблицы баз данных.



В этом задании необходимо написать класс объекта Network, создать на его основе таблицу в БД и добавить в нее пару записей.

Для решения первого упражения создаем базу, потом объект, принимающий её, затем создаем движок базы данных и добавляем к нему объект.

```
Base = declarative_base()

class Network(Base):
     __tablename__ = "network"
     network_id = Column(Integer, primary_key=True)
     name = Column(String(100), nullable=False)

engine = create_engine('sqlite://')
Base.metadata.create_all(engine)
```

```
>>> Base = declarative_base()
>>> class Network(Base):
...     __tablename__ = "network"
...     network_id = Column(Integer, primary_key=True)
...     name = Column(String(100), nullable=False)
...
>>> engine = create_engine('sqlite://')
>>> Base.metadata.create_all(engine)
[SQL]: PRAGMA table_info("network")
[SQL]: ()
[SQL]:
CREATE TABLE network (
        network_id INTEGER NOT NULL,
        name VARCHAR(100) NOT NULL,
        PRIMARY KEY (network_id)
)


[SQL]: ()
[SQL]: COMMIT
```

Затем добавляем экземпляры объектов и делаем коммит сессии.

```
session.add_all([
     Network(name='fred'),
     Network(name='mary'),
     Network(name='wendy')])
session.commit()
```

```
>>> session.add_all([
...     Network(name='fred'),
...     Network(name='mary'),
...     Network(name='wendy')])
>>> session.commit()
[SQL]: BEGIN (implicit)
[SQL]: INSERT INTO network (name) VALUES (?)
[SQL]: ('fred',)
[SQL]: INSERT INTO network (name) VALUES (?)
[SQL]: ('mary',)
[SQL]: INSERT INTO network (name) VALUES (?)
[SQL]: ('wendy',)
[SQL]: COMMIT
```

Следующая часть посвящена выполнению запросов через ORM.

```
+-------------------------------------------------------------------------+
| *** Exercises - ORM Querying ***                                        |
+-------------------------------------------------------------------------+
| 1. Produce a Query object representing the list of "fullname" values for |
|    all User objects in alphabetical order.                              |
|                                                                         |
| 2. call .all() on the query to make sure it works!                      |
|                                                                         |
| 3. build a second Query object from the first that also selects         |
|    only User rows with the name "mary" or "ed".                         |
|                                                                         |
| 4. return only the second row of the Query from #3.                     |
+------------------------------------------------------------ (43 / 72) --+
```

Необходимо создать объект запроса, который предоставляет список полных имен пользователей в алфавитном порядке. Вызвать его полностью методом all(). Затем построить второй запрос, который выберет строки с именем Мэри или Эд, а потом взять вторую запись из него.

```
query1 = session.query(User.fullname).order_by(User.fullname)
print(query1.all())
```

```
>>> query1 = session.query(User.fullname).order_by(User.fullname)
>>> print(query1.all())
[SQL]: SELECT user.fullname AS user_fullname
FROM user ORDER BY user.fullname
[SQL]: ()
[('Ed Jones',), ('Fred Flinstone',), ('Mary Contrary',), ('Wendy Weathersmith',)]
```

```
query2 = session.query(User).filter(or_(User.name=='mary',User.name=='ed'))
print(query2[1]) #т.к. индексируется с 0
```

```
>>> query2 = session.query(User).filter(or_(User.name=='mary',User.name=='ed'))
>>> print(query2[1])
[SQL]: SELECT user.id AS user_id, user.name AS user_name, user.fullname AS user_fullname
FROM user
WHERE user.name = ? OR user.name = ?
 LIMIT ? OFFSET ?
[SQL]: ('mary', 'ed', 1, 1)
<User('mary', 'Mary Contrary')>
```

Следующая часть посвящена мультитабличности.

```
+-------------------------------------------------------------------+
| *** Exercises ***                                                 |
+-------------------------------------------------------------------+
| 1. Run this SQL JOIN:                                             |
|                                                                   |
|    SELECT user.name, address.email_address FROM user              |
|    JOIN address ON user.id=address.user_id WHERE                  |
|    address.email_address='j25@yahoo.com'                          |
|                                                                   |
| 2. Tricky Bonus!  Select all pairs of distinct user names.        |
|    Hint: "... ON user_alias1.name < user_alias2.name"             |
+------------------------------------------------- (62 / 72) --+
```

В этом задании нужно выполнить запрос: вернуть имя пользователя и его адрес из объединения соответсвующих таблиц. Как бонус — вернуть все пары с различными адресами.

```
session.query(User.name, Address.email_address).join(User.addresses).\
       filter(Address.email_address == 'j25@yahoo.com').all()
```

```
>>> session.query(User.name, Address.email_address).join(User.addresses).\
...       filter(Address.email_address == 'j25@yahoo.com').all()
[SQL]: SELECT user.name AS user_name, address.email_address AS address_email_address
FROM user JOIN address ON user.id = address.user_id
WHERE address.email_address = ?
[SQL]: ('j25@yahoo.com',)
[('fred', 'j25@yahoo.com')]
```

```
a1, a2 = aliased(User), aliased(User)
session.query(User).\
         join(a1).\
         join(a2).\
         filter(a1.name < a2.name).\
         all()
```

```
*** Exercises - Final Exam ! ***

1. Create a class called 'Account', with table "account":

      id = Column(Integer, primary_key=True)
      owner = Column(String(50), nullable=False)
      balance = Column(Numeric, default=0)

2. Create a class "Transaction", with table "transaction":
      * Integer primary key
      * numeric "amount" column
      * Integer "account_id" column with ForeignKey('account.id')

3. Add a relationship() on Transaction named "account", which refers
   to "Account", and has a backref called "transactions".

4. Create a database, create tables, then insert these objects:

      a1 = Account(owner='Jack Jones', balance=5000)
      a2 = Account(owner='Ed Rendell', balance=10000)
      Transaction(amount=500, account=a1)
      Transaction(amount=4500, account=a1)
      Transaction(amount=6000, account=a2)
      Transaction(amount=4000, account=a2)

5. Produce a report that shows:
     * account owner
     * account balance
     * summation of transaction amounts per account (should match balance)
       A column can be summed using func.sum(Transaction.amount)
                                                                  (72 / 72)
```

```
from sqlalchemy import Integer, String, Numeric

base = declarative_base()
engine = create_engine("sqlite:///final.db")
connection=engine.connect()
class Account(base):
    __tablename__ = 'account'
    id = Column(Integer, primary_key=True)
    owner = Column(String(50), nullable = False)
    balance = Column(Numeric, default=0)
    def __repr__(self):
        return "<Account(%r, %r)>" % (self.owner, self.balance)
class Transaction(base):
    __tablename__ = 'transaction'
    id = Column(Integer, primary_key=True)
    amount = Column(Numeric, nullable=False)
    account_id = Column(Integer, ForeignKey(Account.__tablename__ + '.id'),
nullable=False)
    account = relationship('Account', backref="transactions")
    def __repr__(self):
        return "Transaction: %r" % (self.amount)
base.metadata.create_all(engine)
session2=Session(bind = eng)
```

```
ac1 = Account(owner='Jack Jones', balance=5000)
ac2 = Account(owner='Ed Rendell', balance=10000)
session.add_all([
    ac1, ac2,
    Transaction(amount=500, account=ac1),
    Transaction(amount=4500, account=ac1),
    Transaction(amount=6000, account=ac2),
    Transaction(amount=4000, account=ac2)])
session.flush()

query = session.query(Account.owner, Account.balance,
func.sum(Transaction.amount)).\
    join(Transaction).\
    group_by(Transaction.account_id)
print(query.all())
```